
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Unidad Iztapalapa

# Práctica #10

**Olvera Monroy Gonzalo**

**<2173011224>**

**Sistemas Operativos**

**Prof. Orlando Muñoz Texzocotetla**

**Trimestre: 20 - O**

## Desarrollo.

## Explicación de los algoritmos usados.

### FIFO:

Uno de los métodos más simples de remplazo de páginas es el método FIFO (First In First Out). Cuando se desea remover una página de la memoria principal, aquella que haya estado en memoria más tiempo será removida.

### Random:

Consiste en elegir un marco aleatoriamente para reemplazar la página.

### FIFO- Segunda Oportunidad:

Consiste en que a cada página en los marcos se le asigna un bit (de oportunidad). El valor del bit va a definir si la siguiente al frente de la cola sale o no. Si el bit de la página al frente de la cola es 0 entonces se reemplaza. Si ese bit vale 1 entonces se reemplaza el siguiente en la cola, y al bit de la página del frente se le cambia el valor a 0.

Cuando se logra entrar a una página a la memoria por primera vez su bit tendrá valor inicial igual a 0. Pero si hay una partición de una página que ya está en memoria entonces cambia su valor del bit a 1.

### Optimo:

En FIFO la página a reemplazar es la primera en entrar. El algoritmo optimo el reemplazo debe de tener el menor índice de fallos de página de todos los algoritmos. En teoría, este algoritmo debe de reemplazar la página que no va a ser usada por el periodo más largo de tiempo.

### LRU:

A cada página se le asocia el tiempo en que fue referenciada, la página elegida por el algoritmo de reemplazo será la que fue accedida hace más tiempo. Es un algoritmo que es óptimo en cuanto al uso hacia atrás de las páginas y es bastante utilizado por los sistemas operativos actuales. No sufre la anomalía de Belady.

La implementación de la práctica se llevó de la siguiente manera:

1. Se crearon funciones para cada algoritmo.
2. Las implementaciones de las funciones fueron con arreglos.
3. En el **main** se implementó para pedir el número de peticiones en total e ingresar la petición una por una y el número de marcos en total. Y después llamo a los métodos para simular la ejecución de los algoritmos **FIFO, FIFO-Segunda Oportunidad, Optimo, LRU y Random.**

```
int main(){

    printf("\nIntroduce el numero de peticiones: ");
    scanf("%d",&num_peticiones);
    int peticiones[num_peticiones];

    printf("\nIntroduce las peticiones:\n");
    for(i = 0; i < num_peticiones; i++){
        scanf("%d", &peticiones[i]);
    }


    printf("\nLas paginas son: \n");
    for ( i = 0;  i< num_peticiones; i++){
        printf(" %d", peticiones[i]);
    }


    printf("\n\nIntroduce el numero de marcos: ");
    scanf("%d",&num_marcos);
    int marcos[num_marcos], tiempo[num_marcos];
    int bit[num_marcos];
    int opcion;

    fifo(peticiones, marcos);
    segunda_oportunidad(peticiones, marcos, bit);
    optimo(peticiones, marcos,tiempo);
    lru(peticiones, marcos, tiempo);
    aleatorio(peticiones, marcos, tiempo);

    return 0;
}
```

- El arreglo **bit[num_marcos]** es para realizar un seguimiento de los bits usados, esa variable la use en la función **segunda_oportunidad.**

- El arreglo **tiempo[num_marcos]** es para realizar un seguimiento del tiempo en que la pagina fue referenciada, la utilizo para los métodos **lru, optimo y aleatorio.**
- En los métodos **lru, optimo y aleatorio** hago **tiempo[j] = cont;** es porque el contenido de **cont** se le asigna al arreglo para tener el tiempo de la ultima referencia de cada página.

Se declararon variables globales, para que todos los métodos las pudieran utilizar.

```
/*
bandera:  tendra valor 1 si la pagina que pide lugar ya esta en memoria
               en otro caso tendra valor 0 y se llevara a cabo un reemplazo
bandera2: tendra valor 1 si la pagina que pide lugar ya esta en memoria
               si no tendra valor de 0 y se podra hacer el reemplazo.
               Tambien es para verificar que no haya marcos y tampoco referenci
a
               a la pagina.
lugar_a_ocupar: indica el indice en arreglo de marcos donde se va a reemplazar
               la pagina correspondiente
i: es el indice para recorrer el arreglo de peticiones
j: es el indice para recorrer el arreglo de marcos
marcos[]:  los marcos donde se agregaran las paginas entrantes
peticiones[]:  va a contener las paginas que piden memoria
*/
int num_peticiones, num_marcos, num_fallas = 0;
int bandera, lugar_a_ocupar = 0;
int i, j, k;
int bandera2;
int cont = 0;
int bandera_imprime_X; //Imprime las X de donde fallo
```

En donde:

- **num_particiones** es el número de peticiones de cuantas se van a hacer.
- **num_marcos** es el número de marcos que se van a ocupar.
- **num_fallas** es para contabilizar cuantas fallas tuvieron cada algoritmo.
- **bandera** va a indicar si hay un reemplazo o no.
- **bandera2** es parecido a la variable bandera la única diferencia es que para los algoritmos **LRU y Random** es el que indica si hay un reemplazo o no.
- **lugar_a_ocupar** indica el numero el índice del arreglo de marcos en donde se va a reemplazar la página correspondiente.
- **i** es el índice para recorrer el arreglo de peticiones.
- **j** es el índice para recorrer el arreglo de marcos.
- **cont** es el índice para recorrer las unidades de tiempo y se incrementa en cada referencia de memoria.
- **bandera_imprime_X** es para imprimir las fallas en pantalla, esta variable la utilice para todos los métodos.

## Implementación para el Método FIFO.

```
/*
El codigo para FIFO lo proporciono el profesor en clase
*/
void fifo(int *peticiones, int *marcos) {
    printf("\n\t      ---FIFO---\n");
    for(i = 0; i < num_marcos; i++)
        marcos[i] = -1;  // -1 significa que ese marco esta desocupado
     num_fallas = 0;

    for(i = 0; i < num_peticiones; i++){ // Se recorre todas las peticiones
        bandera = 0;
        bandera_imprime_X = 0;
        for(j = 0; j < num_marcos; j++){
            if(marcos[j] == peticiones[i])
                bandera = 1; // Si esta la pagina en memoria y entonces NO hay r
eemplazo
                bandera_imprime_X = 0;
        }

        // Si bandera == 0 la pagina entrante no esta en memoria y SI hay reempl
azo
        if(bandera == 0){
            num_fallas++;
            marcos[lugar_a_ocupar] = peticiones[i];
            bandera_imprime_X = 1;
            lugar_a_ocupar = (lugar_a_ocupar+1)%num_marcos;
        }
        //Imprime en donde hay falla

        if(bandera_imprime_X == 1){
            printf("\t X   ");
        } else{
            printf("\t     ");
        }
        for(j = 0; j < num_marcos; j++){
           printf("\t%d ", marcos[j]); // Imprime la memoria en horizontal

        }
        printf("\n");
    } // For principal

    printf("\n\tNumero de fallas Fifo = %d con %d marcos\n",num_fallas, num_marc
os);
```

```
    sleep(2);

}
```

La implementación de la función se llevó de la siguiente manera:

1. El primer **for** es para inicializar cada celda del arreglo **marcos** igual a -1 para indicar que están desocupadas.
2. Lo siguiente se encuentran dos **for** anidados:
    - El primer **for(i = 0; i < num_peticiones; i++)** es para recorrer el número de peticiones, debajo de ese ciclo esta **bandera=0** es para estar actualizando la variable si hay reemplazo o no, si no hay reemplazo la variable es 0.
    - El segundo **for(j = 0; j < num_marcos; j++)** es para recorrer el número de marcos.
        - Debajo del ciclo hay un **if** es para saber si se va a ingresar un numero en la memoria, se tiene que verificar que el número no esté ahí. Y si ya esta se cambia la **bandera** a 1 lo que significa que no hay reemplazo.
    - El siguiente **if** es para la página entrante que no está en memoria y si hay un reemplazo.
        - Dentro de esa condición hay un mensaje que nos indica que página va a ser reemplazada.
        - Se incrementa el número de fallas.
        - Se hace el reemplazo **marcos[lugar_a_ocupar] = peticiones[i]**
    - La variable **bandera_imprime_X** imprime las fallas en la memoria.

## Implementación para el Método Random.

```
/*
Este algoritmo elige al azar una pagina para ser reemplazada
*/
void aleatorio(int *peticiones, int *marcos, int *tiempo) {
    printf("\n\t      ---RANDOM ---\n");
    srand(time(0));

    for(i = 0; i < num_marcos ; i++){
        marcos[i] = -1; // -1 significa que ese marco esta desocupado
    }

    num_fallas = 0;

    for(i = 0; i < num_peticiones; i++){ // Se recorre todas las peticiones
        bandera = 0;
        bandera2 = 0;
        bandera_imprime_X = 0;
```

```
        for(j = 0; j < num_marcos; j++){
            if(marcos[j] == peticiones[i]){ // cuenta cada vez que aparece el nú
mero
                cont++; // Incrementa la unidad de tiempo en el marco
                tiempo[j] = cont;
                bandera = 1; // Si esta la pagina en memoria y entonces NO hay r
eemplazo
                bandera2 = 1;
                bandera_imprime_X = 0;
            }
        }
        if(bandera == 0){ // Si bandera == 0 la pagina entrante no esta en memor
ia y SI hay reemplazo
            for(j = 0; j < num_marcos; j++){
                if(marcos[j] == -
1){  //Si todavia el arreglo de marcos tiene lugares vacios
                    cont++; // Incrementa la unidad de tiempo en el marco
                    num_fallas++; // Incrementa el numero de fallas
                    marcos[j] = peticiones[i];
                    tiempo[j] = cont;
                    bandera2 = 1; //
                    bandera_imprime_X = 1;
                    break;
                }
            }
        }
        int random_posicion;
        if(bandera2 == 0){ // De lo contrario hace el reemplazo
            random_posicion = rand()%num_marcos; // Elige cualquier marco
            lugar_a_ocupar = tiempo[random_posicion];
            cont++; // Incrementa la unidad de tiempo en el marco
            num_fallas++; // Incrementa el numero de fallas
            bandera_imprime_X = 1;
            marcos[lugar_a_ocupar] = peticiones[i];
            tiempo[lugar_a_ocupar] = cont;
        }
        //Imprime en donde hay falla

        if(bandera_imprime_X == 1){
            printf("\t X   ");
        } else{
            printf("\t     ");
        }
        for(j = 0; j < num_marcos; j++){
            printf("\t%d ", marcos[j]); // Imprime la memoria en horizontal
```

```
        }
        printf("\n");
    } // For principal
    printf("\n\tNumero de fallas Random = %d con %d marcos\n",num_fallas, num_ma
rcos);
    sleep(2);
}
```

Esta función lo hice igual que el método **lru** solo con la diferencia que este no busca a la petición menos usada, lo que hace es elegir al azar un marco para reemplazar a una pagina.

1. El primer **for** es para inicializar cada celda del arreglo **marcos** igual a -1 para indicar que están desocupadas.
2. Lo siguiente se encuentran dos **for** anidados:
    - El primer **for(i = 0; i < num_peticiones; i++)** es para recorrer el número de peticiones, debajo de ese ciclo esta **bandera=0 y bandera2=0** es para estar actualizando la variable si hay reemplazo o no, si no hay reemplazo las variables son 0.
    - El segundo **for(j = 0; j < num_marcos; j++)** es para recorrer el número de marcos.
        - Se va incrementando la variable **cont** para contabilizar cuantas veces esta la petición menos usada.
        - Debajo del ciclo hay un **if** es para saber si se va a ingresar un numero en la memoria, se tiene que verificar que el número no esté ahí. Y si ya esta se cambia la **bandera y bandera2** a 1 lo que significa que no hay reemplazo.
        - El siguiente **if(bandera == 0)** es para la página entrante que no está en memoria y si hay un reemplazo.
            - Pero **bandera2** cambia a valor 1 lo que significa que hay marcos y no ha encontrado a la petición que menos se ha usado.
        - Declaro una variable **random_posicion** la cual es para elegir cualquier marco para que se reemplace la petición.
        - El siguiente **if(bandera2 == 0)** es para a hacer el reemplazo de la petición.
            - **random_posicion = rand()%num_marcos** es para elegir cualquier marco.
            - A la variable **lugar_a_ocupar** la igualo al arreglo **tiempo[random_posicion].**
            - Se incrementa **cont** al igual que **num_fallas.**
            - Se hace el reemplazo **marco[lugar_a_ocupar] = peticiones[i]** al igual que **tiempo[lugar_a_ocupar] = cont.**
            - Nos muestra la memoria en pantalla y en donde fallo.

## Implementación para el Método FIFO-Segunda Oportunidad.

```
/*
Este algoritmo consiste en que a cada pagina en los marcos se le asigna un bit (
de oportunidad).
El valor del bit va a definir si la siguiente al frente de la cola sale o no.
*/
void segunda_oportunidad(int *peticiones, int *marcos, int *bit) {
    printf("\n\t    ---FIFO - SEGUNDA OPORTUNIDAD ---\n");

    for (i = 0; i < num_marcos; i++) {
        marcos[i] = -1; // -1 significa que ese marco esta desocupado
        bit[i] = 0; // 0 significa que ese pagina inicia en ese valor
    }

    num_fallas = 0;

    for (i = 0; i < num_peticiones; i++){ // Se recorre todas las peticiones
        bandera = 0; // bandera para las páginas encontradas
        bandera_imprime_X = 0;
        for (j = 0; j < num_marcos; j++){
            if (marcos[j] == peticiones[i]){ // si se encuentra el elemento, use
 el bit a 1 y finalice el ciclo
                bandera = 1; // Si esta la pagina en memoria y entonces NO hay r
eemplazo
                bandera_imprime_X = 0;
                bit[j] = 1;
                break;
            }
        }
        if (bandera == 0){ // si el elemento no se encuentra
            while(bandera != 1){
                // si la memorira todavía tiene lugares vacíos
                if(bit[lugar_a_ocupar] == 0){// si bit = 0 podemos agregar una p
ágina
                    num_fallas++; //incrementa num_fallas
                    bandera = 1; // Si esta la pagina en memoria y entonces NO h
ay reemplazo
                    bandera_imprime_X = 1;
                    bit[lugar_a_ocupar] = 1;
                    marcos[lugar_a_ocupar] = peticiones[i];
                /*
                La memoria está lleno, establece la segunda oportunidad en 0
                para eliminar esa página de la memoria la próxima vez
                */
```

```
                } else {
                    bit[lugar_a_ocupar] = 0;// si no, establece el bit de uso en
 0
                }
                lugar_a_ocupar = (lugar_a_ocupar+1)%num_marcos;
            }
        }
        //Imprime en donde hay falla

        if(bandera_imprime_X == 1){
            printf("\t X   ");
        } else{
            printf("\t     ");
        }
        for(j = 0; j < num_marcos; j++){
           printf("\t%d ", marcos[j]); // Imprime la memoria en horizontal
        }
        printf("\n");
    } // For principal

    printf("\n\tNumero de fallas FIFO - Segunda Oportunidad = %d con %d marcos\n
",num_fallas, num_marcos);
    sleep(2);
}
```

En este método se me facilito usar un arreglo de **bit** que es para ver que a cada página en los marcos se le asigna un bit (de oportunidad) ya que si hubiera utilizado un **struct** tendría que cambiar casi todo mi código y me iba a costar más trabajo para implementar.

**segunda_oportunidad** se implementó de la siguiente manera:

1. El primer **for** es para inicializar cada celda del arreglo **marcos** igual a -1 para indicar que están desocupadas y el arreglo **bit** igual a 0 para que todas las peticiones inicien con un bit en 0.
2. Al igual que el método **FIFO** se encuentran dos **for** anidados.
    - El primer **for(i = 0; i < num_peticiones; i++)** es para recorrer el número de peticiones, debajo de ese ciclo esta **bandera=0 y bandera_imprime_F=0** es para estar actualizando la variable si hay reemplazo o no, si no hay reemplazo la variable es 0.
    - El segundo **for(j = 0; j < num_marcos; j++)** es para recorrer el número de marcos.
        - Debajo del ciclo hay un **if** es para saber si se va a ingresar un numero en la memoria, se tiene que verificar que el número no esté ahí. Y si ya esta se cambia la **bandera** y el arreglo **bit** a 1 lo que significa que no hay reemplazo.

- El siguiente **if(bandera == 0)** es para la página entrante que no está en memoria y si hay un reemplazo.
  - Debajo de esa condición hay un **while(bit != 1)** checa si ninguna petición tiene como **bit** el valor 1.
    - Debajo hay una condición **if(bit[lugar_a_ocupar] == 0)** si tiene el valor de 0 es para ver si la memoria está vacía y se pueda agregar una página.
    - Ya dentro se incrementa el **num_fallas**
    - Se hace el reemplazo **marcos[lugar_a_ocupar] = peticiones[i]**
  - Y si no se estable el **bit** en 0 para poder reemplazarlo más tarde.

## Implementación para el Método Optimo.

```
/*
Este algoritmo debe de reemplazar la página que no va a ser usada por el periodo
 más largo de tiempo.
*/
void optimo(int *peticiones, int *marcos, int *tiempo) {
    printf("\n\t      ---OPTIMO---\n");
    int maximo,bandera3; //bandera3 es la encargada de reemplazar a la pagina qu
e este mas alejado
    for (i = 0; i < num_marcos; i++) {
        marcos[i] = -1; // -1 significa que ese marco esta desocupado
    }

    num_fallas = 0;

    for(i = 0; i < num_peticiones; i++){
        bandera = 0;
        bandera2 = 0;
        bandera_imprime_X = 0;
        // Si la página existe en la memoria
        for(j = 0; j < num_marcos; j++){
            if(marcos[j] == peticiones[i]){
                //Si esta la pagina en memoria y entonces NO hay reemplazo
                bandera = 1;
                bandera2 = 1;
                bandera_imprime_X = 0;
                break;
               }
        }
```

```
        if(bandera == 0){ // si no está en la memoria
            for(j = 0; j < num_marcos; j++){
                if(marcos[j] == -
1){ // si la memoria todavía tiene lugares vacíos
                    num_fallas++;
                    marcos[j] = peticiones[i];
                    bandera2 = 1;
                    bandera_imprime_X = 1;
                    break;
                }
            }
        }
        // si no hay un lugar vacío, entonces la página falla
        if(bandera2 == 0){ // verifica si una página es adecuada para reemplazar
la
            bandera3 = 0;
            // verifica si los números en la memoria aparecerán más tarde o no
            for(j = 0; j < num_marcos; j++){
                tiempo[j] = -1;

                // Se comienza en i + 1, porque vamos hacia adelante, entonces
                // no es necesario comparar a partir de 0, o en i
                for(k = i + 1; k < num_peticiones; k++){
                    if(marcos[j] == peticiones[k]){
                        tiempo[j] = k;
                        break;
                    }
                }
            }

            for(j = 0; j < num_marcos; j++){
                /*
                si no aparece más adelante en la entrada
                guarda su posición para reemplazarla con otra página
                */
                if(tiempo[j] == -1){
                    lugar_a_ocupar = j;
                    bandera3 = 1;
                    bandera_imprime_X = 1;
                    break;
                }
            }

            /*
```

```
            Si todos ellos aparecerán más tarde
            elige uno para eliminarlo de la memoria
            */
            if(bandera3 ==0){
                maximo = tiempo[0];
                lugar_a_ocupar = 0;

                 // es el más lejano, al comparar lugar_a_ocupar con el máximo.
                for(j = 1; j < num_marcos; j++){
                    if(tiempo[j] > maximo){
                        maximo = tiempo[j];
                        lugar_a_ocupar = j;
                    }
                }
            }
            bandera_imprime_X = 1;
            num_fallas++;
            marcos[lugar_a_ocupar] = peticiones[i];
        }

        //Imprime en donde hay falla
        if(bandera_imprime_X == 1){
            printf("\t X   ");
        }
        else{
            printf("\t     ");
        }
        for(j = 0; j < num_marcos; j++){
            printf("\t%d ", marcos[j]); // Imprime la memoria en horizontal
        }
        printf("\n");
    } // For principal

    printf("\n\tNumero de fallas en Optimo = %d con %d marcos\n",num_fallas, num
_marcos);
    sleep(2);
}
```

Para la implementación de la función **optimo** se llevó de la siguiente manera:

1. Declare dos variables:
   - La variable **mayor**
   - La variable **bandera3**
2. El primer **for** es para inicializar cada celda del arreglo **marcos** igual a -1 para indicar que están desocupadas.
3. Lo siguiente se encuentran dos **for** anidados:

- El primer **for(i = 0; i < num_peticiones; i++)** es para recorrer el número de peticiones, debajo de ese ciclo esta **bandera=0 y bandera2=0** es para estar actualizando la variable si hay reemplazo o no, si no hay reemplazo las variables son 0.
- El segundo **for(j = 0; j < num_marcos; j++)** es para recorrer el número de marcos.
    - Debajo del ciclo hay un **if** es para saber si se va a ingresar un numero en la memoria, se tiene que verificar que el número no esté ahí. Y si ya esta se cambia la **bandera y bandera2** a 1 lo que significa que no hay reemplazo.
    - El siguiente **if(bandera == 0)** es por si una página todavía no esta en memoria.
        - **If(marcos[j] == -1)** verifica si la memoria todavía tiene lugares
            - **bandera2** cambia a valor 1 lo que significa que hay marcos y no ha encontrado a la petición que este más alejada.
            - Se incrementa el número de fallas.
    - **If(bandera2 == 0)** verifica si la pagina es la adecuada para ser reemplazada.
        - Debajo de esa condición inicializo la variable **bandera3.**
        - Se encuentra un **for** que es para comprobar si existe una página o no.
        - El siguiente **for(k= i+1; k < num_peticiones; k++)** es para empezar recorrer las peticiones y va contando cuantas veces va apareciendo el número.
        - El siguiente **for** que recorre los **num_marcos** y debajo de ese ciclo hay un **if(tiempo[j] == -1)** es por si no llega aparecer una petición mas adelante entonces guarda su posición para reemplazarla.
    - **If(bandera3 == 0)** es para la pagina entrante, no está en memoria entonces se procede hacer el reemplazo.
        - El **for** es parecido a **LRU** solo con la diferencia de que, en lugar de buscar al menos usado, busca a la petición que este más lejos.
        - Se procede hacer el intercambio **marco[lugar_a_ocupar] = peticiones[i]**
    - Se imprime en pantalla la memoria.

## Implementación para el Método LRU.

```
/*
```

```
En este algoritmo se elige, para reemplazar a la página que más
tiempo lleve sin ser usada.
*/
void lru(int *peticiones, int *marcos, int *tiempo) {
    printf("\n\t        ---LRU---\n");
    int min;

     for(i = 0; i < num_marcos ; i++){
        marcos[i] = -1; // -1 significa que ese marco esta desocupado
    }

     num_fallas = 0;

    for(i = 0; i < num_peticiones; i++){
        bandera = 0;
        bandera2 = 0;
        bandera_imprime_X = 0;
        for(j = 0; j < num_marcos; j++){
            if(marcos[j] == peticiones[i]){ // cuenta cada vez que aparece el nú
mero
                cont++; // Incrementa la unidad de tiempo en el marco
                tiempo[j] = cont;
                bandera = 1; // Si esta la pagina en memoria y entonces NO hay r
eemplazo
                bandera2 = 1; // Si esta en la pagina
                bandera_imprime_X = 0;
            }
        }
        if(bandera == 0){ // Si bandera == 0 la pagina entrante no esta en memor
ia y SI hay reemplazo
            for(j = 0; j < num_marcos; j++){
                if(marcos[j] == -
1){  //Si todavia el arreglo de marcos tiene lugares vacios
                    cont++; // Incrementa la unidad de tiempo en el marco
                    num_fallas++; // Incrementa el numero de fallas
                    marcos[j] = peticiones[i];
                    tiempo[j] = cont;
                    bandera2 = 1; //
                    bandera_imprime_X = 1;
                    break;
                }
            }
        }

        if(bandera2 == 0){ // De lo contrario hace el reemplazo
```

```
            min = tiempo[0];
            lugar_a_ocupar = 0;
            for(j = 1; j < num_marcos; j++){
                if(tiempo[j] < min){
                    min = tiempo[j];
                    lugar_a_ocupar = j;
                }
            }

            //lugar_a_ocupar = encuentra_LRU(tiempo, num_marcos);// Empieza a bu
scar que peticion ha sido menos usada
            cont++; // Incrementa la unidad de tiempo en el marco
            num_fallas++; // Incrementa el numero de fallas
            bandera_imprime_X = 1;
            marcos[lugar_a_ocupar] = peticiones[i];
            tiempo[lugar_a_ocupar] = cont;
        }
        //Imprime en donde hay falla
        if(bandera_imprime_X == 1){
            printf("\t X   ");
        } else{
            printf("\t     ");
        }
        for(j = 0; j < num_marcos; j++){
            printf("\t%d ", marcos[j]); // Imprime la memoria en horizontal
        }
        printf("\n");
    } // For principal
        printf("\n\tNumero de fallas en LRU = %d con %d marcos\n",num_fallas, nu
m_marcos);
        sleep(2);
}
```

Para la implementación de la función **lru** fue de la siguiente manera:

1. El primer **for** es para inicializar cada celda del arreglo **marcos** igual a -1 para indicar que están desocupadas.
2. Lo siguiente se encuentran dos **for** anidados:
   - El primer **for(i = 0; i < num_peticiones; i++)** es para recorrer el número de peticiones, debajo de ese ciclo esta **bandera=0 y bandera2=0** es para estar actualizando la variable si hay reemplazo o no, si no hay reemplazo las variables son 0.
   - El segundo **for(j = 0; j < num_marcos; j++)** es para recorrer el número de marcos.

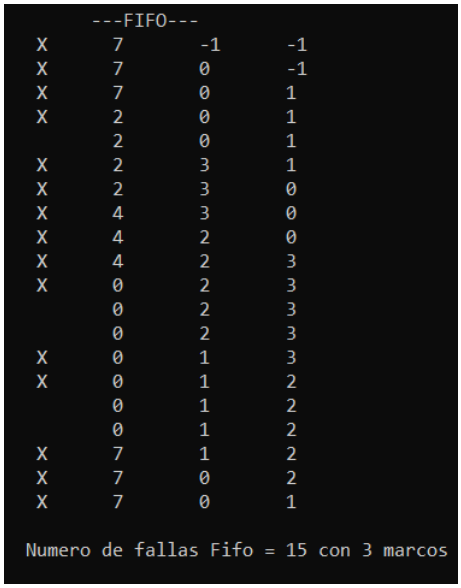
- Se va incrementando la variable **cont** para contabilizar cuantas veces esta la petición menos usada.
- Debajo del ciclo hay un **if** es para saber si se va a ingresar un numero en la memoria, se tiene que verificar que el número no esté ahí. Y si ya esta se cambia la **bandera y bandera2** a 1 lo que significa que no hay reemplazo.
- El siguiente **if(bandera == 0)** es para la página entrante que no está en memoria y si hay un reemplazo.
    - Pero **bandera2** cambia a valor 1 lo que significa que hay marcos y no ha encontrado a la petición que menos se ha usado.
- El siguiente **if(bandera2 == 0)** es para a hacer el reemplazo de la petición con más tiempo que no ha sido usada.
    - El **for** es para recorrer los marcos.
        - Dentro del ciclo hay un **if** que es para verificar si el dato que contiene el arreglo **tiempo** en la posición i es menor lo que contiene la variable **min**, si se cumple entonces se hace el intercambio.
    - Se incrementa **cont** al igual que **num_fallas.**
    - Se hace el reemplazo **marco[lugar_a_ocupar] = peticiones[i]** al igual que **tiempo[lugar_a_ocupar] = cont.**
    - Nos muestra la memoria en pantalla y en donde fallo.

Para todas las funciones **num_fallas** la inicialice en 0, ya que al momento de que se imprime la memoria suma los valores de las fallas del anterior algoritmo ejemplo en **FIFO** salieron 9 fallas, para **FIFO-Segunda-Oportunidad** me da 18 fallas lo cual esta mal porque se empieza a sumar las fallas. Entonces por eso en cada función la inicialice en 0 para no estar sumando las fallas.

Como había mencionado anteriormente la variable **bandera_imprime_X** es para simular a la memoria. La impresión de la memoria la hice en horizontal y que se me dificulto imprimir en memoria como los ejemplos que vienen en las notas.

A continuación, muestro un ejemplo de como se imprime, el ejemplo lo saque de las notas, quería comprobar si imprimía bien las **num_fallas** lo cual lo hace.

**Estas son la paginas: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1**




## 2.- Hacer un análisis comparando los cinco algoritmos para determinar cuál es el que minimice el número de errores. Para esto se deben tomar en cuenta los parámetros:

a. **Número de marcos**
b. **Número de peticiones**
c. **Número de páginas será el doble que el de marcos.**

El algoritmo que minimiza siempre los números de fallos es el **Optimo** ya que estuve probando varios ejemplos que dio el profe, otros buscando en internet y siempre el optimo daba menos fallos.

Por ejemplo, **7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1** estas 20 páginas con 3 marcos.

**FIFO –** 15 fallas

**Segunda-Oportunidad –** 14 fallas

**Optimo –** 9 fallas

**LRU –** 12 fallas

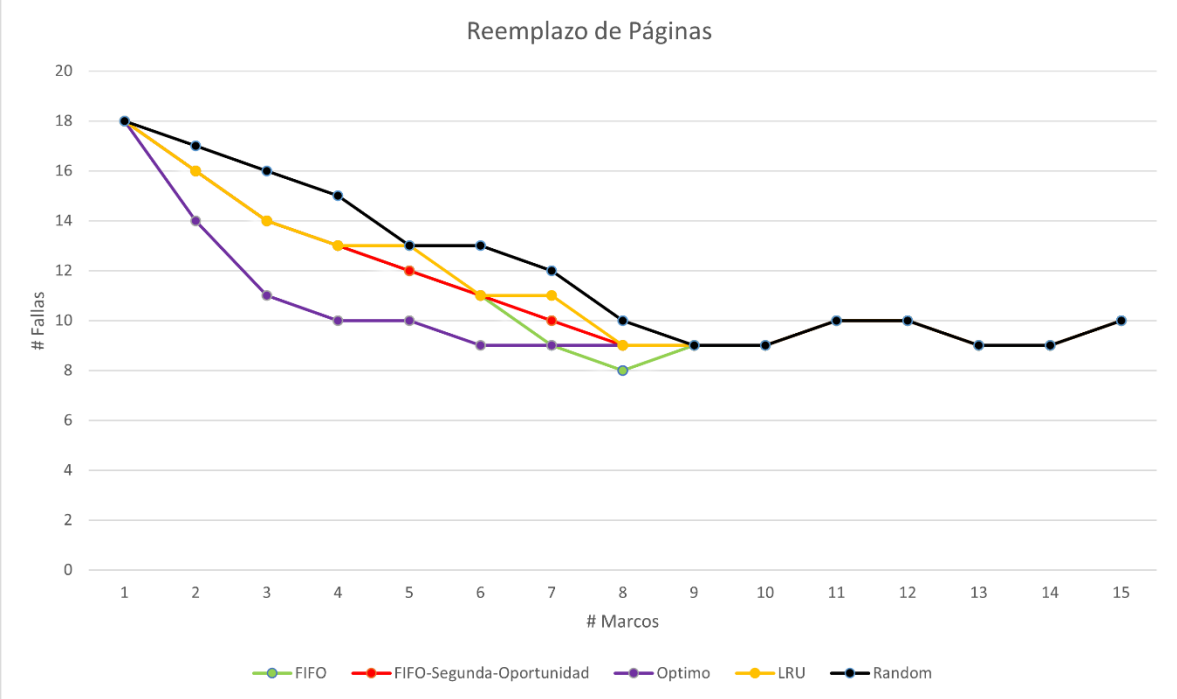
**Random –** 15 fallas

Así fui comparando los algoritmos metiendo ejemplos, y como mencione anteriormente el optimo siempre es que va a tener menos fallas. Pero de lo que me percate es que a veces **FIFO** y **Segunda-Oportunidad** pueden tener el mismo número de fallas, **FIFO** y **Random** muy poco probable es que **Random** tenga menos fallas porque no se sabe que petición puede ser reemplazada y esto ocasiona que **Random** pueda tener menos fallas o más fallas que **FIFO. LRU** a veces tenia menos fallas que **FIFO** y **Segunda-Oportunidad** al igual que **Random.**

Y en lo que me percate fue cuando se tiene 13 marcos y 26 páginas las fallas son las mismas en lo único que cambia a veces es en los intercambios, pero las fallas son las mismas para todos los algoritmos.

A continuación, mostrare la gráfica para ver si se minimizan los errores

Para hacer la gráfica se ejecutó 20 veces el código para obtener el número de fallas de cada algoritmo y sacar el promedio de cada uno con diferentes valores del marco. Del marco #1 al #10 fueron 20 peticiones. Y para los marcos del #11 al #15 solo aumente el doble de páginas.




| FIFO | FIFO-Segunda-Oportunidad | Optimo | LRU | Random |
|---|---|---|---|---|
| 18 | 18 | 18 | 18 | 18 |
| 16 | 16 | 14 | 16 | 17 |
| 14 | 14 | 11 | 14 | 16 |
| 13 | 13 | 10 | 13 | 15 |
| 12 | 12 | 10 | 13 | 13 |
| 11 | 11 | 9 | 11 | 13 |
| 9 | 10 | 9 | 11 | 12 |
| 8 | 9 | 9 | 9 | 10 |
| 9 | 9 | 9 | 9 | 9 |
| 9 | 9 | 9 | 9 | 9 |
| 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 |
| 9 | 9 | 9 | 9 | 9 |
| 9 | 9 | 9 | 9 | 9 |
| 10 | 10 | 10 | 10 | 10 |

Tabla 1. Datos que se obtuvieron con las 20 ejecuciones

**Referencias.**


1. https://www.youtube.com/watch?v=QvoOc79MJ6E
2. https://www.youtube.com/watch?v=ANhMSWCICyQ
3. https://www.youtube.com/watch?v=CgABTUNmofU&t=949s
4. https://nicerova.wordpress.com/2016/10/31/ejercicios-de-algoritmo-de-reemplazo-de-paginas-operating-systems/
5. https://cs.uns.edu.ar/~gd/soyd/clasesgus/10-Memoriavirtual4x.pdf